Mail Aufgabe

January 22, 2018

Die Funktion ‘getNeighbours()‘ möglichst effizient arbeitet und insbesondere keinen Speicher alloziert, d.h. sie sollte vielleicht keinen ‘std::vector¡¿‘ zurückgeben. Ihr könntet etwa ein ‘std::array¡IndexType, MaxPossibleNeighbours¿‘ zusammen mit einer Zahl ‘numActualNeighbours ¡= MaxPossibleNeighbours‘ zurückgeben.

Oder ihr verwendet ein Interface wie bei den STL-Algorithmen, d.h. man übergibt beim Aufruf einen Output-Iterator, der auf einen Container mit Platz für ‘MaxPossibleNeighbours‘ Einträge verweist, und die Funktion gibt den End-Iterator der geschriebenen Sequenz zurück. Es sind sicher auch andere Designs denkbar, die ohne Allokationen auskommen.

Stencil-Operation abbildet: Nehmen wir an, ihr habt einen Vektor x mit N Einträgen, der die Einträge eines d-dimensionalen Meshes repräsentiert. Durch eure Schemaklasse kann man jetzt zu jedem Index iN eine Menge der Nachbarindices $\{n_1^i, \ldots, n_{M_i}^i\}$ erhalten. Einen *averaging stencil* könnte man dann so definieren:

$$y_i = C_i \left(x_i + \sum_{k=1}^{M_i} x_{n_k^i} \right)$$

wobei ich die Gewichtung $C_i := \frac{1}{M_i+1}$ definiert habe. Eine Matrixmultiplikation sieht so aus:

$$y_i = \sum_{j=1}^{N} A_{ij} x_j$$

Zunächst setzen wir nun alle A_{ij} auf 0 (entspricht einer *sparse matrix* mit keinen Einträgen). Für alle Indices i setzen wir dann $A_{ii} = C_i$ und $A_{ij} = C_i$ für alle Nachbarindices $j \in \{n_1^i, \ldots, n_{M_i}^i\}$. Das ergibt eine dünnbesetzte Matrix A_{ij}, und die obige Matrixmultiplikation ist dann dasselbe wie die

Stencil-Auswertung. Ihr könnt also mithilfe eurer Schema-Klasse eine SparseMatrix-Instanz aus Eigen mit Koeffizienten befüllen und dann die Laufzeit für die Multiplikation benchmarken.

Ein Teil eurer Aufgabe sollte außerdem ein Nachweis sein, daß eure Nachbarbestimmungsmethod funktioniert. Z.B. könntet ihr probehalber einen Vektor mit den Indexwerten initialisieren, $x_i = i$, dann einen Stencil anwenden, der den Wert des linken Nachbarn nimmt, und auf dem Ergebnis wieder einen Stencil, der den Wert des rechten Nachbarn nimmt. Dann sollte wieder $x_i = i$ sein, jedenfalls wenn ihr periodische Randbedingungen habt (d.h. der Nachbar eines Punkts am linken Rand ist ein Punkt am rechten Rand).

Noch eine Anmerkung: ich meine mich zu erinnern, daß ihr in eurer Schema-Klasse, die für die Nachbarindexberechnung zuständig ist, auch die Daten selbst speichert (in einem std::vector¡¿ vermutlich). Macht das nicht. Die Indexberechnung und die Datenunterbringung sind getrennte Probleme, und ich möchte eure Klasse vielleicht nicht mit einem std::vector¡¿ verwenden, sondern mit der Vektorklasse aus Eigen, und vielleicht kann/muß ich die gar nicht selber verwalten, sondern ich arbeite nur auf den Daten, die jemand anderes verwaltet.